View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

SCIENCE DIRECT

Discrete Applied Mathematics 154 (2006) 1578–1592

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Improved bottleneck domination algorithms

Ton Kloks[a,1], Dieter Kratsch[b], Chuan-Min Lee[c,*,2], Jiping Liu[a,1,✠]

[a] *Department of Mathematics and Computer Science, The University of Lethbridge, Alta, Canada T1K 3M4*
[b] *Université de Metz, LITA, 57045 Metz, Cedex 01, France*
[c] *Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan*

Received 29 October 2003; received in revised form 7 February 2006; accepted 14 February 2006
Available online 27 March 2006

Abstract

W.C.K. Yen introduced BOTTLENECK DOMINATION and BOTTLENECK INDEPENDENT DOMINATION. He presented an $O(n \log n + m)$-time algorithm to compute a minimum bottleneck dominating set. He also obtained that the BOTTLENECK INDEPENDENT DOMINATING SET problem is NP-complete, even when restricted to planar graphs.

We present simple linear time algorithms for the BOTTLENECK DOMINATING SET and the BOTTLENECK TOTAL DOMINATING SET problem. Furthermore, we give polynomial time algorithms (most of them with linear time-complexities) for the BOTTLENECK INDEPENDENT DOMINATING SET problem on the following graph classes: AT-free graphs, chordal graphs, split graphs, permutation graphs, graphs of bounded treewidth, and graphs of clique-width at most k with a given k-expression.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Bottleneck domination; Bottleneck independent domination; AT-free graphs; Chordal graphs; Permutation graphs

1. Introduction

We consider undirected, finite and simple graphs $G = (V, E)$. We denote by $G[W]$ the subgraph of $G = (V, E)$ induced by the vertex set $W \subseteq V$. For a vertex $x \in V$ in a graph $G = (V, E)$ we use $N(x)$ to denote its neighborhood and $N[x]$ to denote its closed neighborhood $N[x] = N(x) \cup \{x\}$. For $S \subseteq V$, let $N(S) = \bigcup_{s \in S} N(s)$ and $N[S] = \bigcup_{s \in S} N[s]$. Let $w : V \to \mathbb{Z}$ be a function assigning to each vertex v of G a weight $w(v)$ such that all arithmetic operations on vertex weights can be performed in $O(1)$-time. We call w a vertex weight function and (G, w) a weighted graph.

There is a collection of NP-complete graph problems having as input a weighted graph and the goal is to find the minimum (resp. maximum) weight of a certain type of vertex subset, as e.g. MAXIMUM WEIGHT INDEPENDENT SET and MINIMUM WEIGHT DOMINATING SET. Thereby the weight of a vertex set $U \subseteq V$ is usually defined to be $w(U) = \sum_{u \in U} w(u)$. This definition of the weight of a subset determines the objective function of the problem to be considered to be one particular function, namely SUM: minimize (resp. maximize) $\sum_{i=1}^{n} w_i x_i$, where in a standard

✠Upon the completion of this paper the sad news reached us that our friend and coauthor Jiping Liu passed away.

* Corresponding author. Tel.: +886 52720411; fax: +886 52720859.
E-mail addresses: kloks@cs.uleth.ca (T. Kloks), kratsch@lita.univ-metz.fr (D. Kratsch), cmlee@cs.ccu.edu.tw (C.-M. Lee).

[1] Partially supported by NSERC of Canada.
[2] Partially supported by NSC 92-2213-E-194-007.

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.02.003

fashion $V = \{v_1, v_2, \ldots, v_n\}$, and for all $i = 1, 2, \ldots, n$, $w_i = w(v_i)$, and $x_i \in \{0, 1\}$. Although this might seem to be the only interesting objective function, we feel that other objective functions (maybe even nonsymmetric ones) are worth studying. Following earlier work on bottleneck problems we replace the objective function SUM by MAX (resp. MIN):

minimize $\max_{i=1,\ldots,n} w_i x_i$ (resp. maximize $\min_{i=1,\ldots,n} w_i x_i$).

We were motivated by the work of W.C.K. Yen who introduced bottleneck domination and bottleneck independent domination in [22]. He showed that the above approach turns the NP-complete graph problem, (MINIMUM WEIGHT) DOMINATING SET, into the $O(m+n \log n)$-time solvable graph problem BOTTLENECK DOMINATING SET, while (MINIMUM WEIGHT) INDEPENDENT DOMINATING SET is turned into the NP-complete problem BOTTLENECK INDEPENDENT DOMINATING SET (which remains NP-complete on planar graphs). He also gave a linear time algorithm to compute a minimum bottleneck independent dominating set on interval graphs.

Domination is a fundamental concept in graph theory and it also plays an important role as an often studied NP-complete problem. In algorithmic graph theory a huge number of efficient algorithms for the computation of various types of minimum cardinality and minimum weight dominating sets has been published. A special issue of Discrete Mathematics has been published in 1990 [12]. Two books dedicated to domination in graphs have been published in 1998 [10,11].

We recall some notations. A *dominating set* of $G = (V, E)$ is a set D of vertices such that every vertex in $V - D$ has at least one neighbor in D. A *total dominating set* of $G = (V, E)$ is a set D of vertices such that every vertex has at least one neighbor in D (i.e. $G[D]$ has no isolate vertices).

Definition 1. Let (G, w) be a weighted graph. For any set of $V' \subseteq V(G)$, the *bottleneck* of V' is defined to be $\max\{w(x) | x \in V'\}$.

Definition 2. The BOTTLENECK (TOTAL) DOMINATING SET *problem*, B(T)DS, asks for a (total) dominating set D of a weighted graph (G, w) such that the maximum weight over the vertices in D is as small as possible.

The main subject of our paper is the bottleneck variant of the INDEPENDENT DOMINATING SET problem. Recall that an *independent dominating set* of a graph G is a set D of G which is a dominating and an independent set of G.

Definition 3. The BOTTLENECK INDEPENDENT DOMINATING SET *problem* (BIDS) asks for an independent dominating set D of a weighted graph (G, w) such that the maximum weight over the vertices in D is as small as possible.

In a similar way the problems BOTTLENECK CONNECTED DOMINATING SET (where $G[D]$ is connected) and BOTTLENECK DOMINATING CLIQUE (where D is a clique) can be defined.

In our paper we present the following results. In Section 2 we show that there are simple linear time algorithms to compute a minimum bottleneck dominating set and a minimum bottleneck total dominating set. Similar results can be obtained for bottleneck k-tuple domination, bottleneck distance two domination, etc.

The remainder of the paper is dedicated to the BOTTLENECK INDEPENDENT DOMINATING SET problem. In Section 3 we consider the relationship between BIDS and some closely related problems. In Section 4 we present polynomial time algorithms for BIDS on various graph classes. Most of these algorithms have linear time-complexities. In Section 4.1 we show that BIDS can be solved in $O(n^4)$-time for AT-free graphs (a class properly containing the interval graphs). In Section 4.2, we show that this problem can be solved in $O((n+m) \log n)$-time for chordal graphs and in $O(n \log n + m)$-time for split graphs. In Section 4.3 we give a linear time algorithm to solve BIDS on permutation graphs. In Section 4.4 we show that when restricted to graphs of bounded treewidth, BIDS can also be solved in linear time. Finally for graphs of bounded clique-width, in Section 4.5 a linear time algorithm to solve BIDS is given. Assuming that a so-called k-expression is part of the input, we show that there is an $O(5^k k^3 n)$-time algorithm.

Definitions of graph classes and their relevant properties will be given in the corresponding section. For definitions of graph classes or their properties not given in our paper we refer to [4].

Finally let us mention that the bottleneck problems of this paper do not really need weights on the vertices. These weights are only used to induce a linear preference order on the vertices, i.e. the vertex with highest weight is the less wanted. Throughout this paper we shall consider weighted graphs. Clearly if the input graph $G = (V, E)$ is given with a preference order on the vertices this can easily be transformed into a corresponding vertex weight function w.

2. Simple linear time algorithms for BDS and BTDS

The decision variants of our bottleneck problems have a real number η as a part of the input, and the question is asked whether there exists a (total/independent) dominating set with maximum weight (or *bottleneck*) at most η.

Given any real number η it is easy to check whether there exists a dominating, total dominating, connected dominating set D, respectively, with maximum weight at most η: simply take $D_\eta = \{x | w(x) \leqslant \eta\}$. Note that any superset of a dominating, total dominating, connected dominating set, respectively, is also a dominating, total dominating, connected dominating set, respectively. Hence if D_η is dominating, total dominating, connected dominating, respectively, then there exists a dominating, total dominating, connected dominating set of bottleneck at most η, respectively. Otherwise not. In [22] this observation is used to devise a binary search based algorithm that runs in $O(n \log n + m)$-time to solve the BOTTLENECK DOMINATING SET problem.

In this section we show that the BOTTLENECK DOMINATING SET problem can be solved in linear time for all weighted graphs (G, w).

Definition 4. For each vertex x, let $m(x) = \min\{w(u) | u \in N[x]\}$. Let $\rho = \max_{x \in V} m(x)$.

Theorem 1. *The minimum bottleneck of a dominating set in a weighted graph (G, w) is ρ.*

Proof. Consider a dominating set D of G with bottleneck η. Then, since for every vertex x of G its closed neighborhood $N[x]$ has a vertex in D, $m(x) \leqslant \eta$ for all vertices x of G. Hence $\rho \leqslant \eta$. On the other hand, there exists a dominating set with bottleneck ρ. This can be seen by choosing any one vertex y from every closed neighborhood $N[x]$ with $w(y) \leqslant \rho$ in D (such a vertex must exist). Since every closed neighborhood has a vertex in D, this set must be dominating. □

Clearly, ρ can be computed in $O(n + m)$-time, and thus

Corollary 1. *The* BOTTLENECK DOMINATING SET *problem can be solved in $O(n + m)$-time for any weighted graph (G, w).*

We now turn to the BOTTLENECK TOTAL DOMINATING SET problem.

Definition 5. For each vertex x let $m'(x) = \min\{w(u) | u \in N(x)\}$. Let $\rho' = \max_{x \in V} m(x)$.

Theorem 2. *The minimum bottleneck of a total dominating set in (G, w) is ρ'.*

Proof. Consider a total dominating set D with bottleneck η'. Then, since for every vertex x of G its neighborhood $N(x)$ has a vertex in D, for every vertex x of G, $m(x) \leqslant \eta'$ holds. Hence $\rho' \leqslant \eta'$. On the other hand, there exists a total dominating set with bottleneck ρ'. This can be seen by choosing any one vertex y from every neighborhood $N(x)$ with $w(y) \leqslant \rho'$ in D (such a vertex must exist). Since every neighborhood has a vertex in D, this set must be total dominating. □

Corollary 2. *The* BOTTLENECK TOTAL DOMINATING SET *problem can be solved in $O(n + m)$-time for any weighted graph (G, w).*

Our approach can also be applied to the bottleneck version of some generalized domination problems. Let us consider *double domination* introduced in [9]. A double dominating set of a graph $G = (V, E)$ is a set D such that for all $x \in V$, $|N[x] \cap D| \geqslant 2$. (Of course, we need the minimum degree to be at least 1, otherwise such a double dominating set cannot exist.)

To find a bottleneck double dominating set in linear time, we define $m_2(x) = \min_2\{w(y) | y \in N[x]\}$, where $\min_2$ is the smallest but one value in the multiset. Then define $\rho_2 = \max_{x \in V} m_2(x)$. Clearly ρ_2 is the smallest weight of a double dominating set in G. We obtain

Theorem 3. *There is a linear time algorithm to compute a minimum bottleneck double dominating set.*

3. Bottleneck, constrained and minimum weight IDS

The following three problems are closely related to the BOTTLENECK INDEPENDENT SET PROBLEM.

Definition 6. The CONSTRAINED INDEPENDENT DOMINATING SET DECISION problem (CIDS-DECISION) is the following problem:
INSTANCE: A graph $G = (V, E)$ and a set $V' \subseteq V$.
QUESTION: Does G have an independent set $S \subseteq V - V'$ which dominates V', i.e. every vertex of V' has a neighbor in S?

Definition 7. The MINIMUM WEIGHT INDEPENDENT DOMINATING SET problem (WIDS) asks for an independent dominating set D of a weighted graph (G, w) such that the weight of D is minimum, i.e. $\sum_{v\in D} w(v)$ is minimum over all independent dominating sets of G.

Definition 8. The (0–1) INDEPENDENT DOMINATING SET problem ((0–1)IDS) asks for an independent dominating set D of a weighted graph (G, w) such that the weight of D is minimum.

In the rest of this section, we discuss the relationship between BIDS and the above three problems. We also consider the following decision problems corresponding to BIDS, (0–1)IDS, and WIDS.

Definition 9. The BOTTLENECK INDEPENDENT DOMINATING SET DECISION problem (BIDS-DECISION) is the following problem:
INSTANCE: A weighted graph (G, w) and a real number η.
QUESTION: Does G have an independent dominating set S of bottleneck at most η?

Definition 10. The (0–1) INDEPENDENT DOMINATING SET DECISION problem ((0–1)IDS-DECISION) is the following problem:
INSTANCE: A weighted graph (G, w) with 0–1 vertex weight function w.
QUESTION: Does G have an independent dominating set S of weight 0?

Definition 11. The MINIMUM WEIGHT INDEPENDENT DOMINATING SET DECISION problem (WIDS-DECISION) is the following problem:
INSTANCE: A weighted graph (G, w) and a real number k.
QUESTION: Does G have an independent dominating D of weight at most k?

An examination of the proof of Lemma 4 in [22] shows that Yen (who claimed only polynomial equivalence) established the following:

Lemma 1. *The problems, CIDS-DECISION and BIDS-DECISION, are reducible to each other in* $O(n)$*-time.*

His reductions go as follows: Given (G, w, η) as input for BIDS-DECISION, take (G, V') as input for CIDS-DECISION where $v \in V'$ iff $w(v) > \eta$. Given (G, V') as input for CIDS-DECISION, take (G, w, η) as input for BIDS-DECISION where $\eta = 1.5$, and for all $v \in V$, $w(v) = 2$ if $v \in V'$ and $w(v) = 1$ otherwise.

Lemma 2. *BIDS-DECISION can be reduced in* $O(n)$*-time to WIDS-DECISION. Moreover, BIDS-DECISION and* (0–1)*IDS-DECISION are reducible to each other in* $O(n)$*-time.*

Proof. Consider an instance (G, w, η) of BIDS-DECISION. Construct an instance $(G, w', 0)$ of WIDS-DECISION as follows. To each vertex v of G assign weight $w'(v) = 0$ if $w(v) \leqslant \eta$ and $w'(v) = 1$ otherwise. Then there is an independent dominating set of G with bottleneck at most η if and only if (G, w') has an independent dominating set of weight 0. □

Lemma 3. *CIDS-DECISION can be reduced in* $O(n)$*-time to WIDS-DECISION and to* (0–1)*IDS-DECISION.*

Consequently using binary search on the weights of the vertices (and possibly some preprocessing sorting step) BIDS can be solved using WIDS-DECISION or (0–1)IDS-DECISION with an overhead of a $\log n$ factor. CIDS-DECISION can be solved using BIDS without increase of the time bound.

Finally let us emphasize that none of the reductions of this section changes the graph, thus all of the above lemmas remain true when restricted to any graph class.

Theorem 4. *Let $\mathcal{G}$ be any class of graphs. Let $t_B(n,m)$, $t'_C(n,m)$, $t'_{01}(n,m)$ and $t'_W(n,m)$ be the best worst-case running time of an algorithm for BIDS, CIDS-DECISION, (0–1)IDS-DECISION, and WIDS-DECISION on graph class $\mathcal{G}$. Then*

(i) $t_B(n,m) = O(\log n t'_C(n,m))$,
(ii) $t_B(n,m) = O(\log n t'_{01}(n,m))$,
(iii) $t_B(n,m) = O(\log n t'_W(n,m))$,
(iv) $t'_{01}(n,m) = O(t'_W(n,m))$,
(v) $t'_C(n,m) = O(t_B(n,m))$,
(vi) $t'_C(n,m) = O(t'_{01}(n,m))$,
(vii) $t'_C(n,m) = O(t'_W(n,m))$.

Consequently if WIDS, (0–1)IDS or CIDS-DECISION is polynomial time solvable on some graph class $\mathcal{G}$, then BIDS is also polynomial time solvable on $\mathcal{G}$. However it remains possible that WIDS is NP-complete on a graph class $\mathcal{G}'$ while BIDS is polynomial time solvable on $\mathcal{G}'$.

Remark. The problem, CIDS-DECISION, is also introduced in [15] where it is called RED MAXIMAL INDEPENDENT SET. It was shown that CIDS-DECISION can be solved in polynomial time for circle graphs. This is very interesting since it shows that BIDS is polynomial time solvable on circle graphs while even the CARDINALITY INDEPENDENT DOMINATING SET problem remains NP-complete on circle graphs [7]. Thus the complexities of BIDS and the CARDINALITY INDEPENDENT DOMINATING SET problem differ on circle graphs, and this is the only example we know about.

4. Bottleneck independent dominating set

In this section we present algorithms to compute a minimum bottleneck independent dominating set on the following graph classes: AT-free graphs, chordal graphs, split graphs, permutation graphs, graphs of bounded treewidth and graphs of bounded clique-width. Most of these algorithms have linear time-complexities.

4.1. AT-free graphs

In this subsection, we show that the BOTTLENECK INDEPENDENT DOMINATING SET problem can be solved in $O(n^4)$-time for AT-free graphs.

Definition 12. An *asteroidal triple* in a graph G is a set of three vertices in G such there exists a path between every pair of them that avoids the neighborhood of the third. A graph is *asteroidal triple free* (or AT-free) if it contains no asteroidal triple.

The class of AT-free graphs properly contains interval, permutation and cocomparability graphs.
In [5] the following theorem was shown.

Theorem 5. *Let (G,w) be a weighted AT-free graph. Then the* MINIMUM WEIGHT INDEPENDENT DOMINATING SET *problem can be solved in $O(n^4)$-time for (G,w).*

Using Theorem 4 we obtain

Corollary 3. *CIDS-DECISION can be solved in $O(n^4)$-time for AT-free graphs.*

Using Lemma 2 we obtain

Corollary 4. *BIDS-DECISION can be solved in* $O(n^4)$*-time for AT-free graphs.*

By Theorem 4 we also obtain

Corollary 5. *The* BOTTLENECK INDEPENDENT DOMINATING SET *problem can be solved in* $O(n^4 \log n)$*-time for AT-free graphs.*

Since BIDS is a modification of WIDS arising from a change in the objective function it is natural to ask whether any known algorithm for WIDS can be transformed into an algorithm for BIDS by appropriate (and small) changes.

Definition 13. Let $G = (V, E)$ be an AT-free graph, and let x and y be two distinct nonadjacent vertices of G. We use $C^x(y)$ to denote the component of $G - N[x]$ containing y, and $r(x)$ to denote the number of components of $G - N[x]$.

Definition 14. A vertex $z \in V - \{x, y\}$ is *between* x and y if x and z are in one component of $G - N[y]$, and y and z are in one component of $G - N[x]$.

Equivalently, z is between x and y in G if there is an x,z-path avoiding $N[y]$ and there is an y,z-path avoiding $N[x]$.

Definition 15. The *interval* $I = I(x, y)$ of G is the set of all vertices of G that are between x and y.

Definition 16. For a weighted graph (G, w), the minimum weight of an independent dominating set of G is denoted by $\gamma_i^w(G)$.

Broersma et al. [5] propose an $O(n^4)$-time algorithm to solve the MINIMUM WEIGHT INDEPENDENT DOMINATING SET problem for a weighted AT-free graph(G, w) by using dynamic programming on intervals and components of G. All intervals and all components are sorted according to a nondecreasing number of vertices. Following this order, their algorithm computes $\gamma_i^w(C)$ and $\gamma_i^w(C)$ for each component C and each interval I in the order using the formulas given in Lemmas 4–6.

Lemma 4 (*Broersma et al. [5]*). *Let* (G, w) *be a weighted graph,* $G = (V, E)$. *Then*

$$\gamma_i^w(G) = \min_{x \in V} \left\{ w(x) + \sum_{j=1}^{r(x)} \gamma_i^w(C_j^x) \right\},$$

where $C_1^x, C_2^x, \ldots, C_{r(x)}^x$ *are the components of* $G - N[x]$.

Lemma 5 (*Broersma et al. [5]*). *Let* (G, w) *be a weighted AT-free graph,* $G = (V, E)$. *Let* $x \in V$ *and let* C^x *be a component of* $G - N[x]$. *Then*

$$\gamma_i^w(C^x) = \min_{y \in C^x} \left\{ w(y) + \gamma_i^w(I(x, y)) + \sum_j \gamma_i^w(D_j^y) \right\},$$

where the D_j^y*'s are the components of* $G - N[y]$ *contained in* C^x.

Lemma 6 (*Broersma et al. [5]*). *Let* (G, w) *be a weighted AT-free graph,* $G = (V, E)$. *Let* $I = I(x, y)$ *be an interval. If* $I = \emptyset$, *then* $\gamma_i^w(I) = 0$. *Otherwise,*

$$\gamma_i^w(I) = \min_{s \in I} \left\{ w(s) + \gamma_i^w(I(x, s)) + \gamma_i^w(I(s, y)) + \sum_j \gamma_i^w(C_j^s) \right\},$$

where the D_j^s*'s are the components of* $G - N[s]$ *contained in* $I(x, y)$.

Upon closer examination of the methods used in [5] it is clear that the BOTTLENECK INDEPENDENT DOMINATING SET problem can also be solved directly on AT-free graphs in $O(n^4)$-time. The only changes necessary in the formulas of Lemmas 4–6 are changing the summation into a maximization as follows.

Definition 17. For a weighted graph (G, w), the minimum bottleneck of an independent dominating set of G is denoted by $\gamma_b^w(G)$.

Lemma 7. *Let (G, w) be a weighted graph, $G = (V, E)$. Then*

$$\gamma_b^w(G) = \min_{x \in V} \left\{ \max \left\{ w(x), \max_{1 \leqslant j \leqslant r(x)} \gamma_b^w(C_j^x) \right\} \right\},$$

where $C_1^x, C_2^x, \ldots, C_{r(x)}^x$ are the components of $G - N[x]$.

Lemma 8. *Let (G, w) be a weighted AT-free graph, $G = (V, E)$. Let $x \in V$ and let C^x be a component of $G - N[x]$. Then*

$$\gamma_b^w(C^x) = \min_{y \in C^x} \left\{ \max \left\{ w(y), \gamma_b^w(I(x, y)), \max_j \gamma_b^w(D_j^y) \right\} \right\},$$

where the D_j^y's are the components of $G - N[y]$ contained in C^x.

Lemma 9. *Let (G, w) be a weighted AT-free graph, $G = (V, E)$. Let $I = I(x, y)$ be an interval. If $I = \emptyset$, then $\gamma_b^w(I) = -\infty$. Otherwise,*

$$\gamma_b^w(I) = \min_{s \in I} \left\{ \max \left\{ w(s), \gamma_b^w(I(x, s)), \gamma_b^w(I(s, y)), \max_j \gamma_b^w(C_j^s) \right\} \right\},$$

where the D_j^s's are the components of $G - N[s]$ contained in $I(x, y)$.

The correctness, design, and analysis of our algorithm for computing $\gamma_b^w(G)$ of a weighted AT-free graph (G, w) are similar to those of the one for computing $\gamma_i^w(G)$. We refer to [5] for the details. Therefore, we can compute $\gamma_b^w(G)$ in $O(n^4)$-time. It is easy to see that the algorithm for computing $\gamma_b^w(G)$ of a weighted AT-free graph (G, w) can be modified to find an independent dominating set realizing the bottleneck $\gamma_b^w(G)$ with the same time bound. Hence, we have the following theorem.

Theorem 6. *The* BOTTLENECK INDEPENDENT DOMINATING SET *problem can be solved in $O(n^4)$-time on AT-free graphs.*

4.2. Chordal and split graphs

We consider BIDS on chordal and split graphs.

Definition 18. A graph is said to be chordal if it does not contain any cycle of length greater than three as an induced subgraph.

In [8] Farber presents a linear time algorithm to locate a minimum weight independent dominating set in a chordal graph with 0–1 vertex weights based on a linear programming formulation of the problem. Using Theorem 4 we obtain immediately

Theorem 7. *There is an $O((n+m)\log n)$-time algorithm to compute a minimum bottleneck independent set on chordal graphs.*

In [8] Farber also mentions a private communication by Chang saying that WIDS on chordal graphs is NP-complete. Thus chordal graphs are another graph class for which the complexities of BIDS and WIDS differ. In fact the difference in complexity is based on a difference in complexity of WIDS and (0–1)IDS first discovered by Farber.

We consider the following well-known subclass of chordal graphs.

Definition 19. A split graph is a graph $G = (C, S)$, where the vertices of G are partitioned into a clique C and an independent set S.

Without loss of generality we may assume that S is maximal, i.e., every vertex of C has at least one neighbor in S.

First we consider the CONSTRAINED INDEPENDENT DOMINATING SET DECISION problem on split graphs. Let $G=(C, S)$ and $V' \subseteq S \cup C$ be an instance of CIDS. For simplicity, we ask for a maximal independent set without vertices of V'.

Theorem 8. *There exists a maximal independent set in G without any vertex of V' if and only if either S contains only vertices of $V - V'$, or there exists a vertex r of $V - V'$ in C such that all vertices of V' are contained in $N(r)$.*

Proof. Each maximal independent set of G contains at most one vertex of C. Let D be a maximal independent set in G without any vertex of V'. If D contains no vertex of C then $D \subseteq S$, and $N(D) \subseteq C$ implies $D = S$ and $V' \subseteq C$. Otherwise let r be the unique vertex in $D \cap C$. Then all vertices of S being nonadjacent to r must belong to D and thus to $V - V'$. Consequently $V' \subseteq N(r)$. □

Corollary 6. *The* CONSTRAINED INDEPENDENT DOMINATING SET DECISION *problem can be solved in linear time for split graphs.*

By Theorem 4 we obtain immediately that there is an $O((n+m)\log n)$-time algorithm to solve BIDS on split graphs. The following theorem shows that there is a faster algorithm.

Theorem 9. *The* BOTTLENECK INDEPENDENT DOMINATING SET *problem can be solved in* $O(n \log n + m)$*-time for split graphs.*

Proof. First the algorithm sorts the vertices of the independent set S according to non-decreasing weights. Say $S = \{s_1, \ldots, s_\ell\}$ with $w(s_1) \geqslant \cdots \geqslant w(s_\ell)$.

Now, for each vertex $r \in C$, the algorithm determines the minimum index $j(r)$ such that $s_{j(r)} \notin N(r)$ in the following way: for each vertex $r \in C$ the algorithm initializes $j(r) = 1$. Then for $i = 1$ to ℓ it passes through the adjacency list of s_i and for all neighbors $r \in C$ of s_i with $j(r) = i$ it increments $j(r)$.

Let $m(r) = \max\{w(r), w(s_{j(r)})\}$. Clearly, computing $m(r)$ for all $r \in C$ takes $O(n + m)$-time. Finally, compute $\eta = \min_{r \in C} m(r)$.

If $\eta \leqslant w(s_1)$ then η is the minimum bottleneck of an independent dominating set. Otherwise the minimum bottleneck is $w(s_1)$. □

Note that the above algorithm runs in linear time when the graph is given with a preference order on the vertices or when the vertex weight function allows a linear sort of the vertex weights.

4.3. Permutation graphs

In this section we show that there is a linear time algorithm to compute a minimum bottleneck independent dominating set for permutation graphs.

Definition 20. Let π be a permutation of $V_n = \{1, \ldots, n\}$. The *inversion graph* $G(\pi) = (V, E_\pi)$ has vertex set $V = V_n$ and for all $x, y \in V$, $\{x, y\} \in E_\pi$ iff $(x - y)(\pi^{-1}(x) - \pi^{-1}(y)) < 0$. Here $\pi^{-1}(i)$ denotes the position of the number i in the sequence $(\pi(1), \ldots, \pi(n))$. A graph G is a *permutation graph* if there exists a permutation π such that G is isomorphic to the inversion graph $G(\pi)$.

It is convenient to view upon permutation graphs as intersection graphs as follows. Write the number 1 up to n horizontally from left to right. Underneath write the numbers $\pi(1), \ldots, \pi(n)$ also from left to right. Then connect the two 1's, the two 2's etc. by straight line segments. Then two vertices x and y are adjacent whenever the corresponding line segments intersect. This intersection model is called a *permutation diagram*. We label the line segment joining the two i's with label i and in the following we identify the line and its corresponding vertex in the graph $G(\pi)$.

Permutation graphs can be recognized in linear time (and a permutation diagram can be obtained within linear time if the input graph is indeed a permutation graph) [21]. WIDS can be solved in $O(n+m)$-time for permutation graphs [19] (see also [17]).

In this section we show that the linear time algorithm for solving WIDS on permutation graphs can be modified such that the new algorithm solves BIDS on permutation graphs in linear time.

Note that given a permutation diagram of the graph $G=(V, E)$ and a vertex weight function w, then for a line x we could define an immediate successor as a line y such that y lies to the right of x and there is no other parallel line between x and y, i.e. there is no line $z \notin \{x, y\}$ between x and y that crosses neither x nor y.

Definition 21. Any two lines $i, j \in V_n$ with $i \leqslant j$ are a *crossing pair*, denoted by $\mathbb{X}_{ij}$, if $i=j$ or i and j intersect.

For each crossing pair $\mathbb{X}_{ij}$ define $V_{ij}=\{k|k \leqslant j \wedge \pi^{-1}(k) \leqslant \pi^{-1}(i)\}$ and $V'_{ij}=\{k|k<i \wedge \pi^{-1}(k)<\pi^{-1}(j)\}$. Hence, V'_{ij} consists of those lines which lie completely to the left of the lines i and j.

We denote by D_{ij} any independent dominating set of smallest possible maximum weight of $G[V_{ij}]$. Consequently $D_{\pi(n)n}$ is a minimum bottleneck independent dominating set of G.

Definition 22. For a crossing pair $\mathbb{X}_{ij}$, if $V'_{ij}=\emptyset$ then define $\beta(\mathbb{X}_{ij})=\emptyset$. Otherwise let $\beta(\mathbb{X}_{ij})$ be the crossing pair $\mathbb{X}_{i_bj_b}$ satisfying $i_b, j_b \in V'_{ij}$, j_b is the largest number in V'_{ij}, and $\pi^{-1}(i_b) \geqslant \pi^{-1}(k)$ for all $k \in V'_{ij}$.

Hence $\mathbb{X}_{i_bj_b}$ is the crossing pair having j_b as the rightmost top endpoint among all lines in V'_{ij} and having i_b as the rightmost bottom endpoint among all lines in V'_{ij}.

Lemma 10. *Let $j \in V_n$. If $V'_{jj} \neq \emptyset$ then $\beta(\mathbb{X}_{jj})=\mathbb{X}_{i_bj_b}$ and $D_{jj}=D_{i_bj_b} \cup \{j\}$. If $V'_{jj}=\emptyset$ then $\beta(\mathbb{X}_{jj})=\emptyset$ and $D_{jj}=\{j\}$.*

Proof. The vertex j is isolated in $G[V_{jj}]$. Hence any dominating set must contain j. Thus, if $\beta(\mathbb{X}_{jj}) \neq \emptyset$, then $V_{j_bj_b} \cup \{j\}=V_{jj}$. Therefore D is an independent dominating set of $G[V_{jj}]$ if and only if $j \in D$ and $D-\{j\}$ is an independent dominating set of $G[V_{j_bj_b}]$. □

Definition 23. For each crossing pair $\mathbb{X}_{ij}$ with $i<j$ define the clique $\mathscr{C}(\mathbb{X}_{ij})$ as the set of all lines k with $i \leqslant k \leqslant j$ and $\pi^{-1}(j) \leqslant \pi^{-1}(k) \leqslant \pi^{-1}(i)$, and such that there is no line ℓ with $k<\ell<j$ and $\pi^{-1}(k)<\pi^{-1}(\ell)<\pi^{-1}(i)$.

Lemma 11. *Let $\mathbb{X}_{ij}$ be a crossing pair with $i<j$. Then there is a line $k \in \mathscr{C}(\mathbb{X}_{ij})$ such that D_{kk} is an independent dominating set of minimum bottleneck in $G[V_{ij}]$.*

Proof. Let D be an independent dominating set of $G[V_{ij}]$ of smallest maximum weight. Clearly $|D \cap \mathscr{C}(\mathbb{X}_{ij})| \leqslant 1$. Let k be the rightmost line in D. Since D is a dominating set, there cannot be any line ℓ completely to the right of k. Hence $k \in \mathscr{C}(\mathbb{X}_{ij})$. By Lemma 10, D_{kk} is an independent dominating set in $G[V_{ij}]$ of smallest maximum weight and D_{kk} has rightmost line k for some $k \in \mathscr{C}(\mathbb{X}_{ij})$. Hence the bottleneck of D is equal to the bottleneck of D_{kk}. □

Definition 24. Define $\delta(\mathbb{X}_{ii})=i$, and for $i<j$, define $\delta(\mathbb{X}_{ij})$ to be the largest number in $\mathscr{C}(\mathbb{X}_{ij})-\{j\}$.

Hence $\delta(\mathbb{X}_{ij})$ is the line in $\mathscr{C}(\mathbb{X}_{ij})$ with rightmost top endpoint, and its top endpoint is closest to j among all lines of $\mathscr{C}(\mathbb{X}_{ij})$.

Lemma 12. *Let $i<j$ and $t=\delta(\mathbb{X}_{ij})$. Then the smallest maximum weight of an independent dominating set of $G[V_{ij}]$ is equal to the minimum of the maximum weight of D_{jj} and the maximum weight of D_{it}.*

Proof. By Lemma 11, there exists a $k \in \mathscr{C}(\mathbb{X}_{ij})$ such that D_{kk} is a minimum bids of $G[V_{ij}]$. If $k = j$ then D_{jj} is a minimum bids. Otherwise $k < j$ and then $k \in \mathscr{C}(\mathbb{X}_{it}) = \mathscr{C}(\mathbb{X}_{ij}) - \{j\}$ and D_{kk} is an independent dominating set of minimum bottleneck. Consequently D_{it} is a minimum bids of $G[V_{ij}]$. □

It remains to show that $\delta(\mathbb{X}_{ij})$ and $\beta(\mathbb{X}_{ij})$ can be computed in linear time for all crossing pairs $\mathbb{X}_{ij}$. We refer to [19] for these details.

Theorem 10. *There is a linear time algorithm to compute a minimum bottleneck independent dominating set for permutation graphs.*

Proof. Define an ordering $X_{ij} < X_{k\ell}$ if and only if $j < \ell$ or $j = \ell$ and $i > k$. Use the formulas of Lemmas 10 and 12 to compute the minimum bottleneck of an independent dominating set in $G[V_{ij}]$ in increasing order of the crossing pairs $\mathbb{X}_{ij}$ using dynamic programming. Finally, $D_{\pi(n)n}$ is a minimum bottleneck independent dominating set of the graph $G \cong G(\pi)$. □

4.4. Graphs of bounded treewidth

In this section we show that the BIDS problem can be solved in linear time for graphs of bounded treewidth. Let (G, w) be a weighted graph and assume that G has treewidth at most k (for some constant k). The method we describe has already been used for other problems (see e.g. [14]).

It is well-known that when a problem can be formulated in *monadic second order logic* (*MSOL*) then it can be solved in $O(n)$-time for graphs of bounded treewidth [6]. It is easy to see that the INDEPENDENT DOMINATING SET problem can be formulated in MSOL. Hence, WIDS can be solved in $O(n)$-time for graphs of bounded treewidth.

We will present an explicit $O(n)$-time algorithm making use of tree decompositions. Treewidth has originally been defined via tree decompositions of a graph (see e.g. [13]). A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, \mathscr{S})$, where T is a tree and $\mathscr{S} = \{S_i \subseteq V | i \in V(T)\}$ is a collection of subsets of V (called *bags*), one for each node i of the tree T, such that the following three conditions are satisfied. Every vertex $x \in V$ appears in at least one subset S_i. Also, for every edge e in G there is at least one subset S_i containing both endpoints of e. Finally, if a vertex x appears in two bags S_i and S_j then it appears in every bag S_k for k on the (unique) path from i to j in T.

Definition 25. The *width* of a tree decomposition $(T, \mathscr{S})$ (of a graph G) is the maximum cardinality *minus one* over all bags in $\mathscr{S}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G.

It was shown in [3] that for *each* constant k it can be determined in linear time whether a graph G has treewidth at most k.

A tree decomposition $(T, \mathscr{S})$ is *rooted* if the tree T is equipped with some root node. Many problems can be solved efficiently for graphs when a tree decomposition of some constant width is known. For these algorithms it is often useful to make use of a tree decomposition in a certain normalized form. A rooted tree decomposition is called *nice* if every node of T has at most two children and the following conditions are satisfied. If a node i has two children j and k then $S_i = S_j = S_k$ and in this case we call the node i a *join node*. Also, if a node i has only one child j then either $|S_i| = |S_j| + 1$ and $S_j \subset S_i$ (in which case the node i is called an *introduce node*) or $|S_i| = |S_j| - 1$ and $S_i \subset S_j$ (and in this case the node i is called a *forget node*). Finally, if a node i has no children and is not the root, it is called a *start node*.

It is fairly easy to see that every graph with treewidth k has a nice tree decomposition of width k and that it can be obtained in linear time from an ordinary tree decomposition with the same width. Furthermore any graph on n vertices has a nice tree decomposition with at most $4n$ nodes.

By [3,13] we may assume that a nice tree decomposition $(T, \mathscr{S})$ of G of width k is part of the input. For a node i in the tree consider the subtree T_i of T rooted at i. We let G_i stand for the subgraph of G induced by vertices in bags $S_j \in \mathscr{S}$ with j a node of T_i. Our algorithm works from the leaves in T up to the root, computing partial solutions for G_i on the way. These partial solutions are characterized by the status of the nodes in S_i. The status of a node $x \in S_i$ can be of three different types indicated by one of the colors black, grey, and white.

A partial solution for $G_i = (V_i, E_i)$ consists of an independent set $D_i \subseteq V_i$ such that all vertices of $V_i - S_i$ are dominated by vertices of D_i. Furthermore, vertices in $D_i \cap S_i$ are *black*. Vertices of $S_i - D_i$ which are dominated by D_i are colored *grey*. And vertices of $S_i - D_i$ which are not dominated by D_i are *white*.

For a node i in the tree we keep a table of all colorings of the vertices of S_i for which there exists some partial solution D_i for G_i. Furthermore, for each such coloring we also store the minimum maximal weight of a partial solution D_i giving rise to this coloring.

Note that the amount of information stored at each node $i \in T$ is bounded by 3^{k+1} since S_i contains at most $k+1$ vertices. Since k is considered a constant, this is a constant amount of information.

We now describe how to obtain a table of partial solutions for a node i from the tables of its children. We consider the four different cases.

Node **i** *is a start node*: In this case all colorings of vertices in S_i are stored if the black vertices are independent and the grey vertices are exactly those which have a black neighbor. For each such coloring the maximum weight of the black vertices is also stored. If there are no black vertices, this entry is given a value ∞.

Node **i** *is a forget node*: Let $S_i = S_j - \{x\}$. Consider the different colorings in the table stored at node j. If the color of x is *white*, then this can no longer be a partial solution since for partial solutions it is required that all vertices in $V_i - S_i$ are dominated. In all other case, the partial solution is stored as a partial solution in the table at node i, of course without the vertex x. The table at node i is reduced so that it contains no multiple equal entries, i.e., equal except for the value stored. Of course, for the value of such 'equal' entries the minimum is taken.

Node **i** *is a join node*: Let j and h be the two children of i in T. Consider the partial solutions for G_j and G_h characterized by the tables at these nodes. Since $S_i = S_j = S_h$ a coloring for S_i can correspond to a partial solution only if the two sets of black vertices in S_j and S_h are exactly the same. For such corresponding entries at j and h the table at node i contains the *union* of the grey vertices (since a vertex is dominated in G_i if it is dominated in at least one of G_j and G_h). A vertex is white if and only if it is white both in the table entry for j and h. The *value* of this partial solution stored at node i is the maximum of the two values stored at j and h. Again, the table (at node i) is reduced such that it contains no entries which are equal up to the value, and for such 'equal' entries the minimum value is taken.

Node **i** *is an introduce node*: Let $S_i = S_j \cup \{x\}$. Consider again the partial solutions for G_j characterized by the colorings in the table for node j. If x has a black neighbor, then clearly it can only be colored grey. Assume that x has no black neighbor. In that case it is not dominated by the partial solution since all neighbors of x in G_i must be vertices in S_i. We now have two possibilities for creating partial solutions at node i.

The first option is that we color x white. In that case the smallest maximum weight of a partial solution does not change, and neither the color of any vertex in S_i changes.

Secondly, we may choose to color the vertex x black. First of all, we now may have to update the smallest maximal weight of a partial solution with this characteristic if the weight of x exceeds this value. But also, vertices in S_i which were colored white and which are neighbors of x now are colored grey.

Correctness of the procedure described above is evident, and we obtain:

Theorem 11. *For each constant $k \geqslant 1$, there is an $O(5^k n)$-time algorithm to solve BIDS for graphs of treewidth at most k.*

Proof. Consider a join node i with children j and h. For one grey vertex in the table entry for i, one has to compute the minimum value of three entries in j and h: grey and white, white and grey, and grey and grey. Let s be the size of S_i, then there are at most $\sum_{m=0}^{s} \binom{s}{m} 2^{s-m} 3^m = 5^s$ computations to make for the table at node i. This leads to the complexity of $O(5^k n)$-time for this algorithm (see also [1]).

Consider the partial solutions at the root node. Choose the table entry without white vertex with the smallest value attached to it. This corresponds to an independent dominating set of minimum maximal weight.

If we keep pointers at the table entries, telling where the solutions came from, we can also find a minimum bids of G within the same timebound. □

4.5. Graphs of bounded clique-width

In this section we show that for graphs of clique-width at most k, BIDS can be solved in $O(5^k k^3 n)$-time. The method we use is similar to the one used in [16] to obtain an $O(2^{4k} k^2 n)$-time algorithm for the DOMINATION problem.

Graphs of clique-width at most k can be recursively defined by the following four operations on graphs with vertex labels from $\{1, \ldots, n\}$. The clique-width of a graph is the smallest k for which the graph can be constructed by these four operations such that only labels from $\{1, \ldots, k\}$ are used. A *k-expression* is a description of the construction of the graph via these four operations.

- $\mathbf{i(v)}$: The operation creates a new vertex v with label i (incident with no edges).
- $\mathbf{G_1 \oplus G_2}$: The operation takes as input two labelled graphs and creates their disjoint union.
- $\boldsymbol{\eta_{i,j}(G)}$: The operation adds edges between all vertices with label i and all vertices of label j in a labelled graph G.
- $\boldsymbol{\rho_{i\to j}(G)}$: The operation relabels all vertices with label i to label j in a labelled graph G.

Note that there always exists a k-expression with at most $O(k^2n)$ operations (or a corresponding decomposition tree with at most $O(k^2n)$ nodes). This follows since there is no operation *removing* any vertex from a graph.

Definition 26. Let G be a labelled graph with vertex labels from $\{1, \ldots, k\}$. Let S_1, S_2, and S_3 be three *disjoint* subsets of $\{1, \ldots, k\}$. We write $V_G(S_1)$ for the set of vertices of $V(G)$ with a label in S_1. Define $\gamma_G(S_1, S_2, S_3)$ as the minimum bottleneck of an independent dominating set of $G[V(G) - V_G(S_1)]$ which contains no vertex with a label in S_2, and which contains for each label in S_3 at least one vertex with this label. If $V(G) - V_G(S_1) = \emptyset$, we define $\gamma_G(S_1, S_2, S_3) = 0$; otherwise, if such an independent dominating set does not exist we define $\gamma_G(S_1, S_2, S_3) = \infty$.

Note that the minimum bottleneck of an independent dominating set of G is equal to $\gamma_G(\emptyset, \emptyset, \emptyset)$.

For a set S, we use $S + x$ and $S - x$ to denote $S \cup \{x\}$ and $S - \{x\}$, respectively. Now we show how the values $\gamma_G(S_1, S_2, S_3)$ can be computed for each of the four operations.

By the definition, it is not hard to verify Lemmas 13 and 14.

Lemma 13. *Suppose that* $\mathbf{G = i(v)}$. *Then*,

$$\gamma_G(S_1, S_2, S_3) = \begin{cases} 0, & i \in S_1, \\ \infty, & i \in S_2, \\ w(v), & (i \notin S_1 \cup S_2) \wedge ((S_3 = \{i\}) \vee (S_3 = \emptyset)), \\ \infty & \textit{otherwise}. \end{cases}$$

Lemma 14. *Let* G_1 *and* G_2 *be two labelled graphs with vertex labels from* $\{1, \ldots, k\}$. *Suppose that* $\mathbf{G = G_1 \oplus G_2}$. *Then*,

$$\gamma_G(S_1, S_2, S_3) = \min_{S', S'' \subseteq S_3} \left(\max(\gamma_{G_1}(S_1, S_2, S'), \gamma_{G_2}(S_1, S_2, S'')\right),$$

where the minimum is taken over all partitions of S_3 *into two subsets* S' *and* $S'' = S_3 - S'$.

Lemma 15. *Suppose that* $\mathbf{H = \eta_{i,j}(G)}$. *Then*,

$$\gamma_H(S_1, S_2, S_3) = \begin{cases} \infty, & i, j \in S_3, \\ \gamma_G(S_1 + j, S_2 - j, S_3), & i \in S_3 \wedge j \notin S_1 \cup S_3, \\ \gamma_G(S_1 + i, S_2 - i, S_3), & j \in S_3 \wedge i \notin S_1 \cup S_3. \\ \min(\gamma_G(S_1 + i, S_2 - i, S_3 + j), & \\ \quad \gamma_G(S_1, S_2 + j, S_3)) & i \in S_2 \wedge j \notin S_1 \cup S_2 \cup S_3, \\ \min(\gamma_G(S_1 + j, S_2 - j, S_3 + i), & \\ \quad \gamma_G(S_1, S_2 + i, S_3)), & i \notin S_1 \cup S_2 \cup S_3 \wedge j \in S_2, \\ \min(\gamma_G(S_1 + i, S_2, S_3 + j), & \\ \quad \gamma_G(S_1 + j, S_2, S_3 + i), & \\ \quad \gamma_G(S_1, S_2 + i + j, S_3)), & i, j \notin S_1 \cup S_2 \cup S_3, \\ \gamma_G(S_1, S_2, S_3), & i \in S_1 \vee j \in S_1 \vee i, j \in S_2. \end{cases}$$

Proof. We just show the correctness for the following two cases since other cases can be verified in similar ways.

Case 1: $i \in S_3 \wedge j \notin S_1 \cup S_3$. Consider an independent dominating set D that realizes $\gamma_H(S_1, S_2, S_3)$ for this case. D has no vertex with label j and every vertex with label j has at least one neighbor in D. Since any independent dominating set of $H[V(H)-V_H(S_1)]$ without any vertex of label j is also an independent dominating set of $H[V(H)-V_H(S_1+j)]$, therefore $\gamma_H(S_1+j, S_2-j, S_3) \leqslant \gamma_H(S_1, S_2, S_3)$. Let D' be an independent dominating set that realizes $\gamma_H(S_1+j, S_2-j, S_3)$. D' has no vertex with label j and it contains at least one vertex with label i to dominate all vertices with label j. D' is also an independent dominating set of $H[V(H) - V_H(S_1)]$ without any vertex of label j. As discussed above, we have $\gamma_H(S_1, S_2, S_3) = \gamma_H(S_1+j, S_2-j, S_3)$. It is not hard to verify that $\gamma_H(S_1+j, S_2-j, S_3) = \gamma_G(S_1+j, S_2-j, S_3)$. Hence, $\gamma_H(S_1, S_2, S_3) = \gamma_G(S_1 + j, S_2 - j, S_3)$.

Case 2: $i \in S_2 \wedge j \notin S_1 \cup S_2 \cup S_3$. Consider an independent dominating set D that realizes $\gamma_H(S_1, S_2, S_3)$ for this case. D has no vertex with label i. Suppose that D contains at least one vertex with label j. Similar to the proof for Case 1, it is not hard to verify that $\gamma_H(S_1, S_2, S_3) = \gamma_G(S_1 + i, S_2 - i, S_3 + j)$. Suppose that D has no vertex with label j. Clearly $\gamma_H(S_1, S_2, S_3) = \gamma_G(S_1, S_2+j, S_3)$. Hence, we have $\gamma_H(S_1, S_2, S_3) = \min(\gamma_G(S_1+i, S_2-i, S_3+j), \gamma_G(S_1, S_2+j, S_3))$. □

Lemma 16. *Suppose that* $\mathbf{H} = \boldsymbol{\rho}_{\mathbf{i}\to\mathbf{j}}(\mathbf{G})$.

(1) *Assume that* $i \notin S_1 \cup S_2 \cup S_3$. *Then*,

$$\gamma_H(S_1, S_2, S_3) = \begin{cases} \gamma_G(S_1 + i, S_2, S_3), & j \in S_1, \\ \gamma_G(S_1, S_2 + i, S_3), & j \in S_2, \\ \gamma_G(S_1, S_2, S_3 + i), & j \in S_3, \\ \gamma_G(S_1, S_2, S_3), & j \notin S_1 \cup S_2 \cup S_3. \end{cases}$$

(2) *Assume that* $i \in S_1 \cup S_2 \cup S_3$. *Then*,

$$\gamma_H(S_1, S_2, S_3) = \begin{cases} \gamma_H(S_1 - i, S_2, S_3), & i \in S_1, \\ \gamma_H(S_1, S_2 - i, S_3), & i \in S_2, \\ 0, & i \in S_3 \wedge (V(H) - V_H(S_1) = \emptyset), \\ \infty, & i \in S_3 \wedge (V(H) - V_H(S_1) \neq \emptyset). \end{cases}$$

Proof. Note that H has no vertex with label i. Statement (2) can be easily verified according to the definition. For statement (1), we just show the correctness of the following two cases since other cases can be verified analogously. We let $V_j(H)$ (resp. $V_j(G)$) be the set of vertices of $V(H)$ (resp. $V(G)$) with label j and let $V_i(G)$ be the set of vertices of $V(G)$ with label i.

Case 1: $j \in S_1$. Clearly $V_j(H) = V_i(G) \cup V_j(G)$. Hence, $\gamma_H(S_1, S_2, S_3) = \gamma_G(S_1 + i, S_2, S_3)$.

Case 2: $j \notin S_1 \cup S_2 \cup S_2$. In this case i, $j \notin S_1 \cup S_2 \cup S_2$. According to the definition, it is clear that an independent dominating set of G that realizes $\gamma_G(S_1, S_2, S_3)$ can also realize $\gamma_H(S_1, S_2, S_3)$. Hence $\gamma_H(S_1, S_2, S_3) = \gamma_G(S_1, S_2, S_3)$. □

Theorem 12. *For each* $k \geqslant 1$, *there is an* $O(5^k k^3 n)$*-time algorithm to compute a minimum bottleneck independent dominating set of graphs of clique-width at most k assuming the graph is given with a k-expression*.

Proof. The subsets S_1, S_2 and S_3 are disjoint. For the computation of the disjoint union we have to try all $O(2^{|S_3|})$ possible partitions of the set S_3. Note that

$$\sum_{0 \leqslant |S_1|+|S_2|+|S_3| \leqslant k} \binom{k}{|S_1|} \binom{k-|S_1|}{|S_2|} \binom{k-|S_1|-|S_2|}{|S_3|} 2^{|S_3|} = 5^k.$$

There are $O(k^2 n)$ nodes in the decomposition tree. To obtain explicit list of the sets S_i, we need $O(k)$-time per node in the decomposition tree. Finally an independent dominating set of minimum bottleneck can be found using pointers in the standard way. □

Remark. Note that the same method can be used to solve the MINIMUM WEIGHT INDEPENDENT DOMINATION problem. Only a slightly different formulation of the formulas is required.

For some graph classes such as cographs, forests and distance hereditary graphs $O(n+m)$-time algorithms to compute a k-expression (here $k = 2$ resp. $k = 3$) are known.

5. Conclusions

There are simple linear time algorithms for BDS and BTDS. Clearly the BOTTLENECK CONNECTED DOMINATING SET problem can be solved in a straightforward way in $O(m \log n)$-time.

We have presented linear time algorithms for BIDS on various graph classes. The following related questions do seem to be of interest.

Although the best known algorithms for WIDS as well as (Cardinality) IDS on AT-free graphs have complexity of $O(n^4)$-time there might still be a faster algorithm to solve BIDS on AT-free graphs.

Suppose we consider permutation graphs when a permutation diagram is part of the input. In [2] an $O(n \log n)$-time algorithm for WIDS was given under this assumption. Hence, this would lead to an $O(n \log^2 n)$-time algorithm for BIDS on permutation graphs. It would be interesting to know whether there is an $O(n)$-time algorithm for BIDS on permutation graphs (when the permutation diagram is part of the input).

For the related class of trapezoid graphs there exists an $O(n \log n)$-time algorithm for WIDS [20] when a trapezoid diagram is part of the input. Is there an $O(n)$-time algorithm for BIDS on trapezoid graphs (when a trapezoid diagram is part of the input)?

WIDS on cocomparability graphs can be solved by an $O(n^3)$-time dynamic programming algorithm [18]. Breu and Kirkpatrick mention an $O(n^{2.376})$-time algorithm that seemingly they never got published. The $O(n^3)$-time algorithm can easily be turned into an $O(n^3)$-time algorithm for BIDS since we only have to change the objective function for the dynamic programming. Is there a faster algorithm to solve BIDS on cocomparability graphs such as one with $O(n^2)$-time or even $O(n + m)$-time?

The algorithmic complexities of WIDS and BIDS differ for circle graphs and for chordal graphs. It would be nice to find more graph classes for which WIDS or even (0–1)IDS are NP-complete while BIDS is polynomial.

References

[1] J. Alber, H.L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, Fixed parameter algorithms for dominating set and related problems on planar graphs, Algorithmica 33 (2002) 461–493.

[2] M.J. Atallah, S.R. Kosaraju, An efficient algorithm for maxdominance, with applications, Algorithmica 4 (1989) 221–236.

[3] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.

[4] A. Brandstädt, V. Bang Le, J.P. Spinrad, Graph classes—a survey, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.

[5] H. Broersma, T. Kloks, D. Kratsch, H. Müller, Independent sets in asteroidal triple-free graphs, SIAM J. Discrete Math. 12 (1999) 276–287.

[6] B. Courcelle, Graph rewriting: an algebraic and logical approach, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, Elsevier, Amsterdam, 1990, pp. 192–242.

[7] M. Damian-Iordache, S. Pemmaraju, Hardness of approximating independent domination in circle graphs, in: Proceedings of ISAAC 1999, Lecture Notes in Computer Science, vol. 1741, Berlin, 1999, pp. 56–69.

[8] M. Farber, Independent domination in chordal graphs, Oper. Res. Lett. 1 (1981/82) 134–138.

[9] F. Harary, T.W. Haynes, Double domination in graphs, Ars Combin. 55 (2000) 210–213.

[10] T.W. Haynes, S.T. Hedetniemi, P. Slater (Eds.), Fundamentals of Domination in Graphs, Marcel Dekker, New York, 1998.

[11] T.W. Haynes, S.T. Hedetniemi, P. Slater (Eds.), Domination in Graphs: Advanced Topics, Marcel Dekker, New York, 1998.

[12] S.T. Hedetniemi, R.C. Laskar (Eds.), Special volume: topics on domination, Discrete Math. 86 (1–3) (1990).

[13] T. Kloks, Treewidth—Computations and Approximations, Lecture Notes in Computer Science, vol. 842, Springer, Berlin, 1994.

[14] T. Kloks, C.M. Lee, J. Liu, New algorithms for k-face cover, k-feedback vertex set and k-disjoint cycles in plane and planar graphs, in: Proceedings of WG2002, Lecture Notes in Computer Science, vol. 2573, 2002, 282–295.

[15] T. Kloks, C.M. Lee, J. Liu, H. Müller, On the recognition of general partition graphs, in: Proceedings of WG2003, Lecture Notes in Computer Science, vol. 2880, 2003, pp. 273–283.

[16] D. Kobler, U. Rotics, Polynomial algorithms for partitioning problems on graphs with fixed clique-width (extended abstract), in: Proceedings of SODA'01, 2001, pp. 468–476.

[17] D. Kratsch, Algorithms, in: T. Haynes, S. Hedetniemi, P. Slater (Eds.), Domination in Graphs: Advanced Topics, Marcel Dekker, New York, 1998, pp. 191–231, (Chapter 8).

[18] D. Kratsch, L. Stewart, Domination on cocomparability graphs, SIAM J. Discrete Math. 6 (1993) 400–417.
[19] Y.D. Liang, C. Rhee, Linear algorithms for two independent set problems on permutation graphs, Proceedings of the 22nd Computer Science Conference, 1994, pp. 90–93.
[20] Y.L. Lin, Fast algorithms for independent domination and efficient domination in trapezoid graphs, in: Proceedings of ISAAC'98, Lecture Notes in Computer Science, vol. 1533, 1998, pp. 267–276.
[21] R.M. McConnell, J.P. Spinrad, Modular decomposition and transitive orientation, Discrete Math. 201 (1999) 189–241.
[22] W.C.K. Yen, Bottleneck domination and bottleneck independent domination on graphs, J. Inform. Sci. Engrg. 18 (2002) 311–331.